DOCUMENT RESUME

ED 388 276 IR 017 425

AUTHOR Muhlhauser, Max; Rudebusch, Tom
TITLE Cooperation Support in Computer-Aided Authoring and Learning.
PUB DATE 94
NOTE 7p.; In: Educational Multimedia and Hypermedia, 1994. Proceedings of ED-MEDIA 94--World Conference on Educational Multimedia and Hypermedia (Vancouver, British Columbia, Canada, June 25-30, 1994); see IR 017 359.
PUB TYPE Reports - Descriptive (141) -- Speeches/Conference Papers (150)

EDRS PRICE MF01/PC01 Plus Postage.
DESCRIPTORS *Authoring Aids (Programming); *Computer Assisted Instruction; *Computer Software Development; *Cooperative Learning; *Courseware; Educational Environment; Foreign Countries; Intercollegiate Cooperation; Problem Solving; Universities
IDENTIFIERS Germany

ABSTRACT

This paper discusses the use of Computer Supported Cooperative Work (CSCW) techniques for computer-aided learning (CAL); the work was started in the context of project Nestor, a joint effort of German universities about cooperative multimedia authoring/learning environments. There are four major categories of cooperation for CAL: author/author, author/learner, tutor/learner, and learner/learner. In GROUPIE (Group Interaction Environment), a generic support system which was carried out in the Nestor project, a common model and taxonomy of cooperation was established; features include universal development support and a runtime support system. In this system, cooperation is viewed as an aggregate of two lower-level concepts, interaction and coordination. CoopEC (cooperating within the European Community learning domain), an example of CAL-specific courseware, teaches about the EC and features a cooperative problem solving process; this software includes a questionnaire, a multitransition between private and group learning, flexible coupling modes, and both online conferencing and asynchronous document exchange. A more liberal approach to commercial courseware is the authoring system, Nestor-ADP, which consists of cooperation-transparent tools, cooperation-aware tools, and the cooperation-aware lifecycle support environment, DIRECT. In DIRECT, the encompassing graphical user interface supports the cooperative construction of complex, hierarchically structured issue-position argument graphs. In general, two areas of cooperation support are distinguished: author/author cooperation, which involves multiple cooperation-aware tools and learner/side cooperation, which must be supported by generic development support for cooperative courseware. (AEF)

****
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
****

ED 388 276

Cooperation Support in Computer-Aided Authoring and Learning

Max Mühlhäuser and Tom Rüdebusch
University of Karlsruhe, Telecooperation Group
P.O.B. 6980, 76128 Karlsruhe, Germany; [+49] (721) 608-4790

Abstract: This paper discusses the use of Computer Supported Cooperative Work (CSCW) techniques for computer-aided learning (CAL). The work described was started in the context of project Nestor, a large joint effort of german universities and Digital about cooperative multimedia authoring/learning environments (Mühlhäuser and Schaper 1992). We will motivate the specific importance and benefits of CSCW in the CAL domain (chapter 1). Chapter 2 concentrates on a very generic support system, called GROUPIE, which we developed, and shows how it was tailored to CAL; sample cooperative courseware will also be briefly discussed. Chapter 3 covers the second important kind of CSCW support, pre-built cooperative tools, and illustrates their use in a framework for cooperating authors, called DIRECT.

1 Motivation and Classification

The importance of CSCW for the CAL field has been recognized early on, cf. (Wilton 1985; Ward 1991; Smith et al. 1989). As to the authoring side, CAL research in the past years has shown that profound expertise in several disjunct areas is required in order to develop appealing and effective instructional material: *domain knowledge* (about the area to be taught), *instructional design knowledge* (about instructional strategies, learner and knowledge modeling, courseware lifecycles), and *media and interface design expertise*. A single individual can hardly cover all these areas in detail. As a consequence, several people have to cooperate during the process of courseware development in order to obtain reasonable quality. Quite often, such experts work at physically disjunct places. Thus the need for authors to cooperate.

On the learner side, the situation is even more urging: after decades of ITS research, it is recognized that computers can not entirely replace human tutors. As a consequence, learning environments should offer the possibility for the learner to get assistance by knowledgable people (the author, a tutor). Cooperation with authors emphasizes usually on iterative 'feedback loops' that help both the learner (in understanding the subject or the courseware utilization) and the author (in getting hints for improvements). Using more advanced 'liberal' approaches, the distinction between authors and learners even becomes blurred, e.g., if a group of authors tries to acquire knowledge about a domain.

An even more important motivation for cooperation on the learner side is deeply rooted in the modern educational system. The pressure on pupils and students to reach high standards 'produces' rather isolated graduates, accustomed to working alone. Modern media and, not to the least, PCs have largely contributed to this 'lone wolf syndrome'; CAL runs the risk of aggravating this negative trend. At the same time, industry and economy move towards the 'global village'; hardly any reasonable task can be achieved by individuals any more, teamworking skills are strongly required. It is therfore crucial for the success of CAL that courseware addresses cooperation among learners. This goal can only be achieved if teamwork aspects are included in as much courseware material as possible.

To summarize, four major categories of cooperation can be identified for CAL: **author-author, author-learner, tutor-learner, and learner-learner** (note that an individual may play several of these roles in different contexts).

Generic authoring support: interestingly, all but the first one of these categories must be considered in the courseware itself. Since courseware is individually designed for each domain and purpose, pre-built cooperation tools for learners are not very helpful. Early attempts like the provision of electronic mail connections or even videoconferencing have proven to be of limited use. Such approaches require the users to talk *about* the courseware instead of *using* it cooperatively. E.g., for a tutor to provide efficient help, he might need to 'look at' the learner screen, 'ask' the courseware about the learner history, and make remote input to the courseware. Efficient learner cooperation requires that the notion of different learners is 'known' to the courseware and supported. To summarize, generic authoring support for cooperative courseware is crucial in an authoring/learning environment (cf. chapter 2).

Cooperative authoring tools: the first category above, author-author cooperation, is more suited for pre-built cooperative tools than the learner-related scenarios. Chapter 3 will concentrate on this issue in more detail.

IR017425

U.S. DEPARTMENT OF EDUCATION
Office of Educational Research and Improvement
EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)

☐ This document has been reproduced as received from the person or organization originating it.

☐ Minor changes have been made to improve reproduction quality.

• Points of view or opinions stated in this document do not necessarily represent official OERI position or policy.

2

BEST COPY AVAILABLE

"PERMISSION TO REPRODUCE THIS MATERIAL HAS BEEN GRANTED BY

Gary H. Marks

TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)."

2 Generic Authoring Support for Cooperative Software

2.1 Domain-Independent Generic Approach

We investigated many existing cooperative systems in diverse domains, drawing three major conclusions (Rüdebusch 1993). 1: The field of CSCW is lacking a comprehensive and systematic taxonomy; the well-known time-space-categorization (Johansen 1988) is useful but much too restricted. 2: Most existing cooperative applications represent dedicated implementations (wrt. application domain and interaction patterns). 3: All existing CSCW implementations were large programming efforts, again and again realizing similar cooperation functionalities from scratch.

In the GROUPIE (Group Interaction Environment) development which was carried out in the Nestor project, we first established a common model and taxonomy of human-human cooperation in distributed systems. Based on this model, we realized a universal development and runtime support system for cooperative software. Last not least, we used GROUPIE to build example cooperative applications. In GROUPIE, the taxonomy and model of *cooperation* in distributed systems is viewed as an aggregate of two lower-level concepts, namely *interaction* and *coordination*.

A) Interaction: this term denotes any user-initiated action in a CSCW context. It is the basic means of cooperation. Just as actions in a single-user scenario are performed upon objects (e.g. resizing a rectangle in a graphical editor), interactions are performed upon so-called team objects. This new concept extends the well-known object metaphor from single-user work to teamwork. (In the example, resizing the rectangle could be made visible to other group members.) Interactions can have a variety of characteristics, as depicted in fig. 1, "Interaction characteristics".

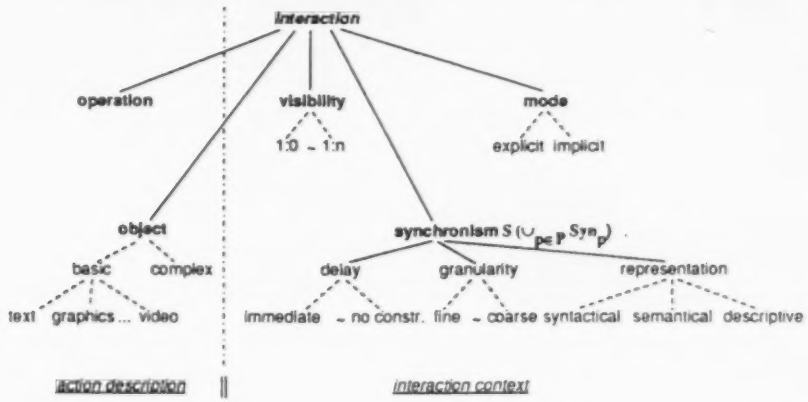

FIGURE 1. Interaction characteristics

In the outset, an actions is defined as usual as an operation performed on a (basic or complex) object. The action description is then augmented to an interaction via the *interaction context*, comprising three components.

The *visibility* component specifies the partners for whom the current interaction is visible (e.g. group members taking part in the cooperative editing of a document). By setting the visibility to 1:0, a single-user action can be specified as a special case of an interaction. (In general, team objects provide a compatible extension of single-user work towards teamwork, as private work is always consistently supported as a special case of group work.)

The *synchronism* component is again subdivided into three components. First, a maximum time delay between the initiation of an interaction and its visibility at a partner can be given. Second, the granularity describes the quantum of sub-actions to be transmitted to a partner (e.g. when resizing a rectangle, each pixel movement, each interval of 10 pixel movements, or only the completed operation can be transmitted). Third, the representation at a partner can be syntactical (exactly the same view), semantical (consistent data but different views, e.g. a pie chart and a bar chart for an integer array), or descriptive (only indicating that an operation has been performed, e.g. 'max has edited figure 1'). Synchronism can be described for each partner individually (e.g. reflecting a tightly cooperating group in a local network with loose connections to a geographically remote member).

The *mode* of an interaction is explicit when a team object is deliberately sent to a partner (e.g. when sending a document via email). Interaction takes place implicitly when acting upon shared team objects (e.g. in a multi-user editor).

B) Coordination: this concept is orthogonal to interaction (i.e. each coordination characteristic can be combined with each interaction characteristic). We speak of basic coordination (or micro coordination) when low-level aspects

BEST COPY AVAILABLE

like authorization and concurrent access are regulated. Complex coordination (or macro coordination) recognizes composite tasks and regulates work flows or conversation structures. In our model, a coordinating framework can be specified (in the beginning independently) for teams and for tasks. In a second step, a team and a task description are combined into a cooperation description. This fosters reusability to a high extent.

Coordination comprises team-, task-, and cooperation-specific aspects. A *team* is described by team-roles and an interaction structure. Team roles abstract from specific users in the team context (e.g. leader, member, protocol-keeper). Interaction structures define interaction paths between team roles (fully democratic, strictly hierarchic, etc.).

A *task* is more complex to describe. Task roles abstract from specific users again, here however as related to the task (e.g. course author, tutor, student). Team objects are the work subject within the task. For each team object, authorization of different task roles and conflict resolution for concurrent access can be specified. Complex tasks can be decomposed into sub-tasks recursively. Constraints watch over sequences of sub-tasks, temporal and logical conditions.

A *cooperation* associates a team with a common task by mapping team-relevant and task-relevant roles.

Coordination takes place on an interaction granularity, i.e. each interaction during task performance is checked against the coordination descriptions. Coordination can also influence task performance actively by guiding team members (e.g. suggesting a next sub-task to work on, based on the coordination descriptions).

Development support in the GROUPIE system comprises a library of team objects, formal languages (for team, task, and cooperation description) and cooperative tools and methods. Thus, CSCW functionality is implemented on an object granularity, as opposed to the common approach of tool granularity. Teams and tasks are not strictly predefined but can dynamically be adapted to changing needs. A cooperative method for developing CSCW systems in teams has been implemented and can be reused or further refined (bootstrap approach).

The runtime system: GROUPIE *distribution support* provides a high-level object-oriented interface based on (home-brewn) Distributed Smalltalk. *Distribution management* decides about migration, replication, and consistency. Cooperative applications are embraced by (guiding and checking) *coordination support*, *interaction handling* interprets and realizes an interaction context. It is possible to integrate existing single-user applications in a cooperation-transparent way (for the term cooperation-transparent, cf. chapter 3).

2.2 CAL-Specific Extensions to GroupIE and Sample Courseware

According to our concept of generic support for cooperative software as introduced above, adaptation to specific application domains such as CAL is technically simple. Existing base classes need to be adequately refined. Much more difficult is the question of "what are adequate team objects and strategies for cooperative courseware?".

Cooperative problem solving can, in a first approach, be supported by simply providing an information space to the learners that is freely accessible. Information about the learning domain can be retrieved both cooperatively and individually if all information objects are realized as team objects (e.g. documents, video sequences). *Student evaluation* requires dedicated learning objects (e.g. questionnaire, multiple choice documents for cooperative completion); tutors (with specific privileges) may be involved. *Cooperative strategies* are even more interesting: well-known learning strategies for individuals (drill&practice, tutorial, exploratory learning, etc.) must be complemented by cooperative ones which, e.g., take into account different roles of learners. E.g., procedural knowledge of legislation in the European Community may be taught based on various roles of delegates in the Commission; a cooperative strategy for information acquisition is described in ch. 3; a motivational approach might use the paradigm of 'multi-user dungeons', etc.

coopEC (cooperating within the European Community learning domain) is as sample courseware for validation of our concepts. coopEC teaches about the EC and features a cooperative problem solving process and working on various documents. Four basic team objects specific to CAL have been developed in the first version: a questionnaire, a multiple choice document, a map of Europe with hot spots, and a free-form text/graphics document. It supports dynamic transition between private and group learning, flexible coupling modes, and both on-line conferencing and asynchronous document exchange. Once enrolled,students can also be sent documents while logged off.

Fig. 2 shows the main window of coopEC. The sub-window at the top is used for cooperation control. The very detailed set of possible interaction characteristics as introduced in chapter 2.1 has been customized into a few user-selectable combinations, each accessible via one respective button, that seemed most appropriate for this specific setting.

The user can, at any time, choose 'private' work or cooperate with his partners. To cooperate, he can 'share' documents (with the whole group or a sub-group) or 'send' a document to any number of partners. The closeness of a cooperation can be determined by selecting one of three 'coupling' buttons. With 'tight' coupling in share mode, all partners have the same view on the shared document. Any operation, like typing a single character, is visible for each

of the partners. Full telepointing is supported, as the movement of each partners' mouse pointer is overlaid onto the shared document. In send mode, tight coupling results in a mail window automatically being brought on top of all other windows for each receiver. 'Medium' coupling in share mode transmits only completed interactions. E.g. when filling out the questionnaire, only complete answers are transmitted to the partners. 'Loose' coupling, whether in share or send mode, just prints a textual information about the operation performed in the message area at the very bottom of the main window. The sub-window below cooperation control provides meta information about users and modes. coopEC supports two roles: tutor and learner (indicated in brackets). Users listed as inactive have never accessed the course. A unique color is mapped to each user. This is extremely helpful when sharing a document (to distinguish multi-user input and telepointers). Green is reserved for the tutor, making him easy to identify.

Four buttons 'questionnaire', 'multiple choice', 'euro map', and 'note' are provided to select a document to work on (details skipped for the sake of brevity). As an example scenario, a learner could decide to start a cooperative questionnaire by answering the questions he already knows. He may then invite some of the enrolled students to share this partly completed questionnaire.The sub-group can then cooperate by inserting or correcting answers (the color of each answer text dynamically changes to the unique color of the current editor). Telepointers can be used to indicate issues currently in discussion. The learners may decide to involve the tutor and finally submit the questionnaire.

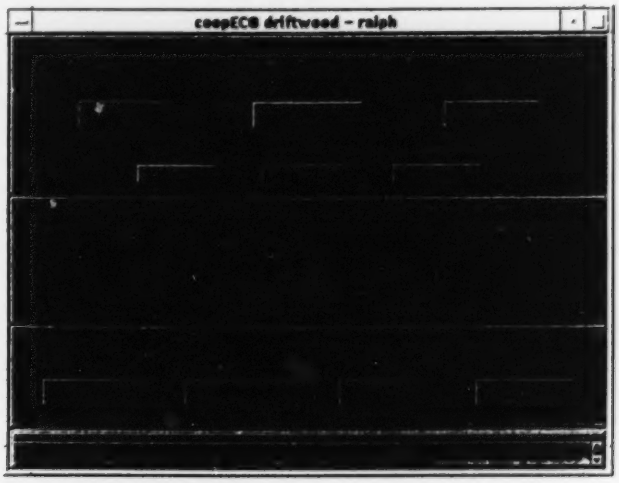

FIGURE 2. Top-level user interface for coopEC

3 Cooperative Authoring Tools and Lifecycle

Commercial courseware development has in the past been based on rather prescriptive lifecycles. New management theories and cooperation trends suggest more liberal approaches, as does the academic working style. We will concentrate on such a more liberal approach in the remainder (more prescriptive lifecycles can easily be supported). The resulting authoring system Nestor-ADP is comprised of cooperation-transparent tools, cooperation-aware tools, and the cooperation-aware lifecylce support environment DIRECT.

Cooperation transparent tools: this term refers to tools which have been built with a single user in mind. The idea is to extend them transparently for use from multiple user interfaces, possibly in a distributed system (the advantage that existing tools can be augmented is paid with limited cooperation functionality).

Nestor tool "shX": In the context of our project Nestor, generic cooperation support for cooperation-transparent software has been developed, called "shX". shX allows prebuilt Xwindow™ -based application software to be used from multiple workstations (even without recompilation or rebinding). All output from the application is replicated to all participating workstations. shX offers different so-called floor passing schemes, ranging from 'prescriptive token-based' (only one user has control at a time) to 'anarchy mode' (uncontrolled multiple input). New schemes may be added, such as role-specific ones (e.g., 'tutor-learner', where the tutor always has superior input rights).

IDE-shX: since Nestor tries to support the entire courseware lifecycle, we built a specific tool for the early courseware design phase, called 'instructional design editor'. This tool allows to arrange instructional goals and objec-

BEST COPY AVAILABLE

tives graphically into a semantic network, relating these instructional issues to domain knowledge. IDE 'understands' several instructional strategies. IDE has been extended by shX in order to allow cooperative courseware design.

CSE-shX: IDE-based designs can be 'compiled' with one of the instructional strategies implemented in IDE, generating template course structures for the 'course structure editor' CSE. CSE is based on a high-level graphical notation for the flow of control and actions in the courseware, called 'instructional transactions. The template generated (by IDE) for CSE provides the courseware skeleton which has to be complemented by concrete screen and media designs and by fine-tuning for the instructional transactions. A cooperative shX-based extension of CSE has been built. In the long run, we plan to implement cooperation-aware successor tools to IDE and CSE based on GROUPIE.

Cooperation-aware tools: as mentioned above, cooperation-transparent applications are very restricted. E.g, they can hardly be used if teams work together at different times. Since the courseware lifecycle is a long-living activity, such asynchronous cooperation is however crucial. Therefore, we had to build several cooperation-aware tools plus a cooperation-aware lifecycle framework encompassing the (cooperation-transparent and -aware) tools. In addition, integration of audio and video conferencing support was found to be an absolute requirement for efficient cooperation of physically distributed authoring teams.

GROUPIE itself is, for the moment, our most important cooperation-aware environment. In addition, the following cooperation-aware project management tools were built: a meeting coordination tool which interworks with private calendar tools and with the other tools listed here; a group mail system (which uses the user's standard mail for sending/receiving mail); individual calendar tools; a time / task planner based on critical path networks CPN. Apart from that, tools for audio and video conferencing have been implemented, featuring dynamic conference management. The audio conferencing is based on our low-cost SCSI™-based hardware audio extension for workstations. The video conferencing support uses vendor-supplied frame grabbing boards and a home-brewn software video codec, called SMP (Neidecker-Lutz and Ulicheny 1993).

Cooperation-aware lifecycle support: at a first glance, one may decide to base a cooperative framework for courseware lifecycle integration on traditional courseware development processes. However, it was already argued in chapter 2 that modern views of cooperative work tend to seek new, more liberal management and organization approaches than the model of subsequent "phases" (like 'define', 'design', 'develop', 'deliver',...). In fact, if courseware development should lead to new interesting solutions, the outcome and goals are not very clear at the outset: courseware development becomes an 'ill-structured problem'. An excellent approach to coping with such ill-structured problems has been presented in (Potts 1989), called 'issue-based design'. In our project, we adapted this concept to the problem space of courseware development and implemented a cooperative framework accordingly. The framework was called 'DIRECT' (distributed research and engineering for cooperating CAL teams).

The courseware-adapted issue-based design approach shall be briefly described here. According to this approach, the problems associated with developing a specific courseware are structured into 'issues' (each issue can be hierarchically separated into sub-issues). For each issue (or sub-issue), different approaches may be identified over time (these approaches are called 'positions' according to the original method), and arguments supporting or objecting to an approach may be collected in the cooperating group of authors. In DIRECT, the encompassing graphical user interface supports the cooperative construction of complex, hierarchically structured issue-position-argument graphs. A particular value of the system stems from the provision of authoring-specific issue-position-argument templates which are predefined for reuse.

According to the method, the authoring team has to opt for one of the alternative approaches to an issue. At this point, a number of possible so-called 'steps' are offered to the authors which they can choose from in order to resolve the issue (an example for a standard step is the development of a courseware module). It is (mainly) here that the above-cited tools are 'hooked' into the encompassing framework: if a specific step is chosen, a kind of 'workflow support system' is started which manages the coordinated use of the above-mentioned courseware design/development tools, project management tools and audio/video conferencing tools. Our experience has shown that this integral approach has a lot of advantages over the uncoordinated use of individual cooperative tools. As an example, if a step starts with the definition of a task force (e.g., assigned for the development of a specific courseware module), then subsequent tools can be automatically provided with the names and network addresses of the relevant group participants.

The screenshot in fig. 3 shows the screen of a DIRECT user, a partial display of an issue-position-argument graph (upper right) and some cooperative tools which are actually in use by the author (e.g., IDE-shX at the lower left).

4 Conclusions

We have shown that two basic areas of cooperation support have to be distinguished, author-author cooperation and

learner-side cooperation. Learner-side cooperation must be supported by generic development support for cooperative courseware rather than by tools. Author-author cooperation involves multiple cooperation-aware (and maybe cooperation-transparent) tools and can be best accomplished with the help of an encompassing framework. The latter was described based on a liberal courseware lifecycle. Stable versions of GROUPIE, coopEC, cooperative authoring tools and DIRECT as described are actually running on our premises and at several external sites.

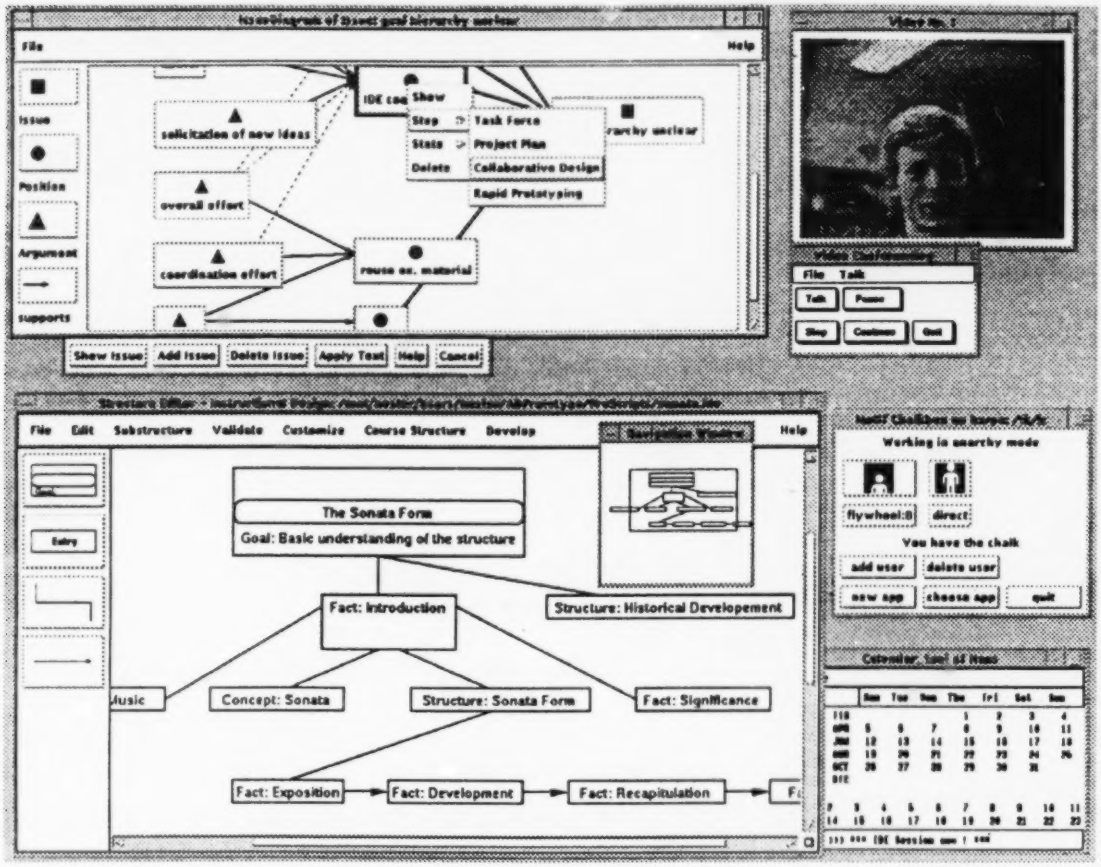

FIGURE 3. Sample Screenshot of the DIRECT Courseware Lifecycle Framework

5 References

Johansen, R. (1988): Groupware: Computer Support for Business Teams. *The Free Press, New York.*

Mühlhäuser, M., Schaper, J. (1992): Project NESTOR: New Approaches to Cooperative Multimedia Authoring/ Learning. *Tomek, I. : Computer Assisted Learning, Springer Berlin, 453-465.*

Neidecker-Lutz, B., Ulicheny, R. (1993): Software Motion Pictures. *Digital Technical Jl. 5, 19-27.*

Potts, C. (1989): A generic model for representing design methods. *in: Druffel, L. (Ed.): Proc. 11th Intl. Conf. on Software Engineering, Pittsburgh, PA. ACM Press, New York, 217-226.*

Rüdebusch, T. (1993): CSCW - Generic Support for Teamwork in Dist. Systems. *DUV Wiesbaden,* in German.

Smith, R.B. et al. (1989): Preliminary Experiments with a Distributed, Multi-Media Problem Solving Environment. *in: Proc. ECSCW 1st Europ. Conf. on Comp.Supported Coop. Work, 19-34*

Ward, D.R. (1991): Boosting Connectivity in a Student Generated Collaborative Database. *in: Proc. ECSCW 2nd Europ.Conf. on Comp.Supported Coop.Work, Kluwer, Amsterdam, 191-201.*

Wilton, J.A. (1985): Structuring Learning Networks. *in: WCCE, World Conf. on Computers in Education, 491-496.*

BEST COPY AVAILABLE